CDAT Utilities 3.3 – CHAPTER 2

CHAPTER 2

General Utilities : The genutil Package

The functions in the genutil package are written to be general purpose functions that are useful to a broader community and not restricted to climate data applications.

Statistics Functions

Statistics functions available in this package include commonly used functions to compute correlation, covariance, auto−correlation, auto−covariance, lagged correlation, lagged covariance, mean absolute difference, root mean square, standard deviation, variance, geometric mean, median, percentiles and linear regression.

correlation

Returns the correlation between 2 slabs. By default on the first dimension, centered and biased by default.

Usage:

result = correlation(x, y, weights=*weightoptions*, axis=*axisoptions*, centered=*centeredoptions*, biased=*biasedoptions*)

Options:

weightoptions

default = None. If you want to compute the weighted correlation, provide the weights here.

axisoptions `x' | `y' | `z' | `t' | `(dimension_name)' | 0 | 1 ... | n

default value = 0. You can pass the name of the dimension or index (integer value 0...n) over which you want to compute the statistic.

centeredoptions None | 0 | 1

default value = 1 removes the mean first. Set to 0 or None for uncentered.

biasedoptions None | 0 | 1

default value = 1 returns biased statistic. If want to compute an unbiased statistic pass anything but 1.

Example:

```
# Let us try an example where we want to look at a

# variable `tas' from the NCEP reanalysis and compute

# some spatial statistics between data slices for time

# periods from 1960−1970 and 1980−1990.

>>> import cdms

>>> from genutil import statistics

>>> f = cdms.open('tas.rnl_ncep.nc')

>>> ncep1 = f('tas',time=(`1960−1−1', `1970−1−1', 'co'))

>>> ncep2 = f('tas',time=(`1980−1−1', `1990−1−1', 'co'))

# We have the two time periods extracted.

# Now let us compute the correlation.

>>> cor = statistics.correlation(ncep1, ncep2,\

axis='xy')

# We could compute the spatial correlation weighted by

# area. To accomplish this we can use the `generate'

# option for weights.

>>> wcor = statistics.correlation(ncep1, ncep2,\

weights='generate', axis='xy')
```

covariance

Returns the covariance between 2 slabs. By default on the first dimension, centered and biased by default.

Usage:

cov = covariance(x, y, weights=*weightoptions*, axis=*axisoptions*, centered=*centeredoptions*, biased=*biasedoptions*)

Options:

weightoptions

default = None. If you want to compute the weighted covariance, provide the weights here.

axisoptions `x' | `y' | `z' | `t' | `(dimension_name)' | 0 | 1 ... | n

default value = 0. You can pass the name of the dimension or index (integer value 0...n) over which you want to compute the statistic.

centeredoptions None | 0 | 1

default value = 1 removes the mean first. Set to 0 or None for uncentered.

biasedoptions None | 0 | 1

default value = 1 If want to compute an unbiased variance pass anything but 1.

autocorrelation

Returns the autocorrelation of a slab at lag k centered,partial and "biased" by default

Usage:

result = autocorrelation(x, lag=*lagoptions*, axis=*axisoptions*, centered=*centeredoptions*, partial=*partialoptions*, biased=*biasedoptions*, noloop=*noloopoptions*)

Options:

lagoptions None | n | (n1, n2, n3...) | [n1, n2, n3 ....]

default value = None the maximum possible lags for specified axis is used.You can pass an integer, list of integers, or tuple of integers.

axisoptions `x' | `y' | `z' | `t' | `(dimension_name)' | 0 | 1 ... | n

default value = 0. You can pass the name of the dimension or index (integer value 0...n) over which you want to compute the statistic.

centeredoptions None | 0 | 1

default value = 1 removes the mean first. Set to 0 or None for uncentered

partialoptions None | 0 | 1

default value = 1 uses only common time for means.

biasedoptions None | 0 | 1

default value = 1 computes the biased statistic. If want to compute an unbiased statistic pass anything but 1.

noloopoptions None | 0 | 1

default value = 0 computes statistic at all lags upto `lag'. If you set noloop=1 statistic is computed at lag only (not up to lag).

autocovariance

Returns the autocovariance of a slab. By default over the first dimension, centered, and partial.

Usage:

result = autocovariance(x, lag=*lagoptions*, axis=*axisoptions*, centered=*centeredoptions*, partial=*partialoptions*, noloop=*noloopoptions*)

Options:

lagoptions None | n | (n1, n2, n3...) | [n1, n2, n3 ....]

default value = None the maximum possible lags for specified axis is used.You can pass an integer, list of integers, or tuple of integers.

axisoptions `x' | `y' | `z' | `t' | `(dimension_name)' | 0 | 1 ... | n

default value = 0. You can pass the name of the dimension or index (integer value 0...n) over which you want to compute the statistic.

centeredoptions None | 0 | 1

default value = 1 removes the mean first. Set to 0 or None for uncentered

partialoptions None | 0 | 1

default value = 1 uses only common time for means.

noloopoptions None | 0 | 1

default value = 0 computes statistic at all lags upto `lag'. If you set noloop=1 statistic is computed at lag only (not up to lag).

laggedcorrelation

Returns the correlation between 2 slabs at lag k centered, partial and "biased" by default.

Usage:

result = laggedcorrelation(x,y, lag=*lagoptions*, axis=*axisoptions,* centered=*centeredoptions*, partial=*partialoptions*, biased=*biasedoptions*, noloop=*noloopoptions*)

Returns value for x lags y by lag

Options:

lagoptions None | n | (n1, n2, n3...) | [n1, n2, n3 ....]

default value = None the maximum possible lags for specified axis is used.You can pass an integer, list of integers, or tuple of integers.

axisoptions `x' | `y' | `z' | `t' | `(dimension_name)' | 0 | 1 ... | n

default value = 0. You can pass the name of the dimension or index (integer value 0...n) over which you want to compute the statistic.

centeredoptions

default value = 1 removes the mean first. Set to 0 or None for uncentered

partialoptions None | 0 | 1

default value = 1 uses only common time for means.

biasedoptions None | 0 | 1

default value = 1 If want to compute an unbiased variance pass anything but 1.

noloopoptions None | 0 | 1

default value = 0 computes statistic at all lags upto `lag'. If you set noloop=1 statistic is computed at lag only (not up to lag).

laggedcovariance

Returns the covariance between 2 slabs at lag k centered and partial by default

Usage:

result = laggedcovariance(x, y, lag=*lagoptions*, axis=*axisoptions*, centered=*centeredoptions*, partial=*partialoptions*, noloop=*noloopoptions*)

Returns value for x lags y by lag (integer)

Options:

lagoptions None | n | (n1, n2, n3...) | [n1, n2, n3 ....]

default value = None the maximum possible lags for specified axis is used.You can pass an integer, list of integers, or tuple of integers.

axisoptions `x' | `y' | `z' | `t' | `(dimension_name)' | 0 | 1 ... | n

default value = 0. You can pass the name of the dimension or index (integer value 0...n) over which you want to compute the statistic.

centeredoptions

default value = 1 removes the mean first. Set to 0 or None for uncentered

partialoptions None | 0 | 1

default value = 1 uses only common time for means.

noloopoptions None | 0 | 1

default value = 0 computes statistic at all lags upto `lag'. If you set noloop=1 statistic is computed at lag only (not up to lag).

meanabsdiff

Returns the mean absolute difference between 2 slabs x and y. By default on the first dimension and centered

Usage:

result = meanabsdiff(x, y, weights=*weightoptions*, axis = *axisoptions*, centered=*centeredoptions*)

Options:

weightoptions

default = None returns equally weighted statistic. If you want to compute the weighted statistic, provide weights here.

axisoptions `x' | `y' | `z' | `t' | `(dimension_name)' | 0 | 1 ... | n

default value = 0. You can pass the name of the dimension or index (integer value 0...n) over which you want to compute the statistic.

centeredoptions None | 0 | 1

default value = 1 removes the mean first. Set to 0 or None for uncentered.

Example:

To compute the mean absolute difference between ncep1

and ncep2.

>>> absd = statistics.meanabsdiff(ncep1, \ ncep2,axis='xy')

rms

Returns the root mean square difference between 2 slabs. By default from a slab (on first dimension) "uncentered" and "biased" by default

Usage:

result = rms(x, y, weights=*weightoptions*, axis = *axisoptions*, centered=*centeredoptions,* biased = *biasedoptions*)

Options:

weightoptions

default = None returns equally weighted statistic. If you want to compute the weighted statistic, provide weights here.

axisoptions `x' | `y' | `z' | `t' | `(dimension_name)' | 0 | 1 ... | n

default value = 0. You can pass the name of the dimension or index (integer value 0...n) over which you want to compute the statistic.

centeredoptions None | 0 | 1

default value = 0 returns uncentered statistic (same as None). To remove the mean first (i.e centered statistic) set to 1. NOTE: Most other statistic functions return a centered statistic by default.

biasedoptions None | 0 | 1

default value = 1 If want to compute an unbiased variance pass anything but 1.

Example:

To compute the "temporal" rms difference between the

two time periods

>>> rms = statistics.rms(ncep1, ncep2, axis='t')

std

Returns the standard deviation from a slab. By default on first dimension, centered, and biased.

Usage:

result = std(x, weights=*weightoptions*, axis = *axisoptions*, centered=*centeredoptions*, biased = b*iasedoptions*)

Options:

weightoptions

If you want to compute the weighted statistic, provide weights here.

axisoptions `x' | `y' | `z' | `t' | `(dimension_name)' | 0 | 1 ... | n

default value = 0. You can pass the name of the dimension or index (integer value 0...n) over which you want to compute the statistic.

centeredoptions None | 0 | 1

default value = 1 removes the mean first. Set to 0 or None for uncentered.

biasedoptions None | 0 | 1

default value = 1 If want to compute an unbiased variance pass anything but 1.

variance

Returns the variance from a slab. By default on first dimension, centered, and biased.

Usage:

result = variance(x, weights=*weightoptions*, axis = *axisoptions*, centered=*centeredoptions*, biased = *biasedoptions*)

Options:

weightoptions

If you want to compute the weighted variance, provide weights here.

axisoptions `x' | `y' | `z' | `t' | `(dimension_name)' | 0 | 1 ... | n

default value = 0. You can pass the name of the dimension or index (integer value 0...n) over which you want to compute the statistic.

centeredoptions None | 0 | 1

default value = 1 removes the mean first. Set to 0 or None for uncentered.

biasedoptions None | 0 | 1

default value = 1 If want to compute an unbiased variance pass anything but 1.

geometricmean

Returns the geometric mean over a sepcified axis.

Usage:

result = geometricmean(x, axis=*axisoptions*)

Options:

axisoptions `x' | `y' | `z' | `t' | `(dimension_name)' | 0 | 1 ... | n

default value = 0. You can pass the name of the dimension or index (integer value 0...n) over which you want to compute the statistic.

percentiles

Returns values at the defined percentiles for an array.

Usage:

result = percentiles(x, percentiles=*percentilesoptions*, axis=*axisoptions*)

Options:

percentilesoptions A python list of values

Default = [50.] (the 50th percentile i.e the median value)

axisoptions 'x' | 'y' | 'z' | 't' | '(dimension_name)' | 0 | 1 ... | n

default value = 0. You can pass the name of the dimension or index (integer value 0...n) over which you want to compute the statistic.

median

Returns the median value of an array.

Usage:

result = median(x, axis=*axisoptions*)

Options:

axisoptions 'x' | 'y' | 'z' | 't' | '(dimension_name)' | 0 | 1 ... | n

default value = 0. You can pass the name of the dimension or index (integer value 0...n) over which you want to compute the statistic.

linearregression

Computes the linear regression of y over x or an axis. This function returns Values of the slope and intercept, and optionally, Error estimates and associated probability distributions for T−value (T−Test) and F−value (for analysis of variance f) can be returned. You can choose to return all these for either slope or intercept or both (default behaviour). For theoretical details, refer to "Statistical Methods in Atmospheric Sciences" by Daniel S. Wilks, Academic Press, 1995.

Usage:

result = linearregression(y, axis=*axisoptions*, x=*xvalues*, error=*erroroptions*, probability=*probabilityoptions*, nointercept=*nointerceptoptions*, noslope=*noslopeoptions*)

Options:

axisoptions `x' | `y' | `z' | `t' | `(dimension_name)' | 0 | 1 ... | n

default value = 0. You can pass the name of the dimension or index (integer value 0...n) over which you want to treat the array as the dependent variable.

xvalues

default = None. You can pass an array of values that are to be used as the independent axis x

nointerceptoptions None | 0 | 1

default = None. Setting to 0 or None means intercept calculations are returned. To turn OFF the intercept computations set nointercept to 1.

noslopeoptions None | 0 | 1

default = None. Setting to None or 0 means slope calculations are returned. To turn OFF the slope computations set noslope to 1.

erroroptions None | 0 | 1 | 2 | 3

default = None. If set to 0 or None, no associated errors are returned.

If set to 1, the unadjusted standard error is returned.

If set to 2, standard error returned. This standard error is adjusted using the centered autocorrelation of the residual.

If set to 3, standard error returned. The standard error here is adjusted using the centered autocorrelation of the raw data (y).

probabilityoptions None | 0 | 1

default = None. If set to 0 or None, no associated probabilities are returned. Set this to 1to compute probabilities.

Note: Probabilities are returned only if erroroptions are set to one of 1, 2, or 3. If it is set to None or 0, then setting probabilityoptions has no meaning.

What is returned?

The returned values depend on the combination of options you select. If both slope and intercept are required, a tuple is returned for both Value and optionally Error (or optionally associated Probabilities), but single

values (not tuples) are returned if only one set (slope OR intercept) is required. See examples below for more details.

When *erroroption* = 1 (from description above for *erroroptions* you know that means unadjusted standard error) and *probabilityoption* = 1, then the following are returned:

pt1 : The p−value for regression coefficient t−value. (With no adjustment for standard error or critical t−value.)

None : There is only one p−value to be returned (pt1) but None is returned to keep the length of the returned values consistent.

pf1 : The p−value for regression coefficient F−value (one−tailed).

pf2 : The p−value for regression coefficient F−value (two−tailed).

When *erroroption* = 2 or 3 (implying error adjustment using the residual or the raw data and *probabilityoption* = 1, then the following are returned:

pt1 : The p−value for regression coefficient t−value.(With Effective sample size adjustment for standard error of slope.

pt2 : The p−value for regression coefficient t−value.(With effective sample size adjustment for standard error of slope and critical t−value.)

pf1 : The p−value for regression coefficient F−value (one−tailed).

pf2 : The p−value for regression coefficient F−value (two−tailed).

The values pt1 and pt2 are used to test the null hypothesis that b = 0 (i.e., y is independent of x). The values pf1 and pf2 are used to test the null hypothesis that the regression is linear (goodness of linear fit). For non−replicated values of y, the degrees of freedom are 1 and n−2.

The xmgrace module

Nothing emphases the fact that CDAT is a collection of tools that can be extended by the user better than the ***xmgrace*** module. This module provides an interface to the popular Grace plotting utility (which you must have installed separately. Downloads and information are available from http://plasma−gate.weizmann.ac.il/Grace).

The tutorials (see the document Climate Data Analysis Tools (CDAT): A beginner's Guide or the ***CDAT*** home page at http://cdat.sf.net for details) include two tutorials that demonstrate the use of python in getting full use out of XmGrace.

Additional convenience functions

minmax

Returns the minimum and maximum of a series of arrays/lists/tuples (or a combination of these). You can combine list/tuples/... pretty much any combination is allowed.

Examples of Use:

```
>>> import genutil

>>> s = range(7)

>>> genutil.minmax(s)

(0.0, 6.0)

>>> genutil.minmax([s,s])

(0.0, 6.0)

>>> genutil.minmax([[s,s*2],4.,[6.,7.,s]],\

[5.,7.,8,(6.,1.)])

(−7.0, 8.0)
```

grower

This function takes 2 transient variables and grows them to match their axes.

Usage:

x, y = grower(x, y, singleton=*singletonoption*)

Options:

singletonoption 0 | 1

Default = 0 If singletonoption is set to 1 then an error is raised if one of the dims is not a singleton dimension.

rgb2str

Given r,g,b values, this function returns the closest 'name'

Example:

```
>>> print rgb2str([0,0,0])

'black'
```

str2rgb

Given a string representing a color name, this function the corresponding r,g,b values (between 0 and 255). If the color name is unknown, the function returns None,None,None

This is accomplished by looking in the /usr/X11R6/lib/X11/rgb.txt file. If the file does not exist, then looks into the builtin dictionary

Examples:

```
>>> r, g, b = str2rgb('pink2')

# returns: (238 , 169 , 184 )



>>> r, g, b = str2rgb('crappy')

# returns: (None, None, None)
```

Go to Main | Go to Previous | Go to Next